BLOCK PRECONDITIONERS BASED ON APPROXIMATE COMMUTATORS [*]

HOWARD ELMAN[†], VICTORIA E. HOWLE[‡], JOHN SHADID[§], ROBERT SHUTTLEWORTH[¶], AND RAY TUMINARO[‖]

Abstract. This paper introduces a strategy for automatically generating a block preconditioner for solving the incompressible Navier–Stokes equations. We consider the "pressure convection–diffusion preconditioners" proposed by Kay, Loghin, and Wathen [11] and Silvester, Elman, Kay, and Wathen [16]. Numerous theoretical and numerical studies have demonstrated mesh independent convergence on several problems and the overall efficacy of this methodology. A drawback, however, is that it requires the construction of a convection–diffusion operator (denoted F_p) projected onto the discrete pressure space. This means that integration of this idea into a code that models incompressible flow requires a sophisticated understanding of the discretization and other implementation issues, something often held only by the developers of the model. As an alternative, we consider automatic ways of computing F_p based on purely algebraic considerations. The new methods are closely related to the "*BFBt* preconditioner" of Elman [6]. We use the fact that the preconditioner is derived from considerations of commutativity between the gradient and convection–diffusion operators, together with methods for computing sparse approximate inverses, to generate the required matrix F_p automatically. We demonstrate that with this strategy, the favorable convergence properties of the preconditioning methodology are retained.

Key words. Preconditioning, Navier–Stokes, iterative algorithms

AMS subject classifications. Primary, 65F10, 65N30; Secondary, 15A06

1. Introduction.

Recently, the development of efficient iterative methods for the fully-implicit solution of the Navier–Stokes equations has seen considerable activity. In this paper, we consider a promising class of methods (denoted *pressure convection–diffusion* preconditioners) proposed by Kay, Loghin, and Wathen [11] and Silvester, Elman, Kay, and Wathen [16]. These preconditioners are designed to solve linear systems associated with the incompressible Navier–Stokes equations

$$\begin{array}{rcl} \alpha \mathbf{u}_t - \nu \nabla^2 \mathbf{u} + (\mathbf{u} \cdot \mathrm{grad})\, \mathbf{u} + \mathrm{grad}\, p & = & \mathbf{f} \\ -\mathrm{div}\, \mathbf{u} & = & 0 \end{array} \quad \text{in } \Omega \subset \mathbb{R}^3, \tag{1.1}$$

where $\mathbf{u}$ are the velocities, p are the pressures, and $\mathbf{u}$ satisfies suitable boundary conditions on $\partial\Omega$. The value $\alpha = 0$ corresponds to the steady-state problem and

[*]This work was partially supported by the DOE Office of Science MICS Program and by the ASCI Program at Sandia National Laboratories under contract number DE-AC04-94AL85000. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

[†]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, elman@cs.umd.edu. The work of this author was supported by the National Science Foundation under grant DMS0208015, and by the Department of Energy under grant DOEG0204ER25619.

[‡]Sandia National Laboratories, PO Box 969, MS 9159, Livermore, CA 94551, vehowle@sandia.gov.

[§]Sandia National Laboratories, PO Box 5800, MS 1111, Albuquerque, NM 87185, jnshadi@cs.sandia.gov.

[¶]Applied Mathematics and Scientific Computing Program, University of Maryland, College Park, MD 20742, rshuttle@math.umd.edu.

[‖]Sandia National Laboratories, PO Box 969, MS 9159, Livermore, CA 94551, rstumin@sandia.gov.

$\alpha = 1$ to the transient case. Linearization and stable discretization (discussed in Section 2) yield systems of equations of the form

$$\begin{pmatrix} F & G \\ D & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ g \end{pmatrix} \tag{1.2}$$

where G and D are discrete gradient and (negative) divergence operators, respectively, and F contains all other terms coming from the linearized momentum equation and time derivatives. These systems, which are the focus of this paper, must be solved at each step of a nonlinear (Picard or Newton) iteration or at each time step.

The pressure convection–diffusion preconditioner is based on an approximate block factorization of (1.2). The key to attaining mesh independent convergence lies with the effective approximation of the inverse of the following operator:

$$S = DF^{-1}G. \tag{1.3}$$

S is the Schur complement of (1.2) and is obtained by algebraically eliminating the velocity equations. Most effective block preconditioners for (1.2) must approximate the inverse of the Schur complement in some fashion. The presence of the term F^{-1} between the rectangular operators D and G makes S prohibitively expensive to compute and almost completely dense. The pressure convection–diffusion preconditioner can be viewed as a way to approximate (1.3) by an expression in which the inverse term is moved (or commuted) so that it no longer appears between the two rectangular operators. This approximation takes the form

$$S \approx \hat{S} = DGF_p^{-1} \tag{1.4}$$

where F_p is a discrete convection–diffusion operator defined on the discrete pressure space. This operator will be discussed in detail in Section 2.

This idea requires the explicit construction of the matrix F_p. In all previous work on these schemes, this operator has been constructed using a discretization consistent with that used for the pressure term of the Navier–Stokes equations. For example, if linear finite elements are used to discretize the pressures in (1.1), then F_p is defined using the same basis. Although most codes used to model incompressible flow have the primary kernels needed for such a construction, this places a significant burden on a potential user of this methodology, something not found in standard algebraic preconditioners.

Our aim in this study is to rectify this situation, and to enable the preconditioning methodology derived from (1.4) to be available to users in "black-box" fashion requiring nothing more than a statement of the problem (1.2). This is achieved using the property of commutativity alluded to above. Specifically, we derive an approximate solution to the algebraic problem

$$GF_p \approx FG \tag{1.5}$$

where F_p must be determined given G and F. This is accomplished by considering a least-squares approximation to (1.5). We show how this leads to a natural interpretation of the related "*BFBt* algorithm"[6]. We also illustrate how the above commutator equation can be modified to incorporate diagonal scaling and how proper scaling is critical to obtaining satisfactory convergence rates. In one algebraic variant, sparse approximate inverse ideas are adapted to the above commutator strategy. We present this approach in Section 4, where we give a specific algorithm based on the methods of Grote and Huckle [9]. In Section 5, numerical results are given to illustrate the competitiveness of the automatically computed pressure convection–diffusion preconditioner with the original version of this approach [11, 16], (which requires the user to provide the matrix F_p) in terms of convergence speed.

2. The pressure convection–diffusion preconditioner. We are concerned with the incompressible form of the Navier–Stokes equations given by (1.1). Our focus is on solution algorithms for the systems of equations that arise after linearization of the system (1.1). We will restrict our attention to a Picard iteration for the nonlinear system, derived by lagging the convection coefficient in the quadratic term $(\mathbf{u} \cdot \mathrm{grad})\,\mathbf{u}$. For the steady-state problem, this lagging procedure starts with some initial guess $\mathbf{u}^{(0)}$ (satisfying the discrete incompressibility constraint) for the velocities and then computes the kth Picard iterate for velocity and pressure by solving the *Oseen equations*

$$\begin{aligned} -\nu\nabla^2\mathbf{u}^{(k)} + (\mathbf{u}^{(k-1)} \cdot \mathrm{grad})\,\mathbf{u}^{(k)} + \mathrm{grad}\, p^{(k)} &= \mathbf{f} \\ -\mathrm{div}\,\mathbf{u}^{(k)} &= 0. \end{aligned} \tag{2.1}$$

For transient problems, a strategy of this type can be combined with an implicit time discretization, see [17, 18].

A stable finite difference, finite volume, or finite element discretization of (2.1) leads to a linear system of equations of the form (1.2) which must be solved at each step of the Picard iteration. For the steady problem, the matrix F has block diagonal form in which each individual diagonal block consists of a discretization of a convection–diffusion operator

$$-\nu\nabla^2 + (\mathbf{w} \cdot \mathrm{grad})\,, \tag{2.2}$$

where $\mathbf{w} = \mathbf{u}^{(k-1)}$. For the transient problem, the blocks of F represent discretizations of an operator essentially of the form

$$\frac{1}{\Delta t} I - \nu\nabla^2 + (\mathbf{w} \cdot \mathrm{grad})\,, \tag{2.3}$$

which arises from an implicit time discretization of the time-dependent convection–diffusion equation.

The strategy we employ for solving (1.1) is derived from the block factorization

$$\begin{pmatrix} F & G \\ D & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ DF^{-1} & I \end{pmatrix} \begin{pmatrix} F & G \\ 0 & -S \end{pmatrix}, \tag{2.4}$$

where $S = DF^{-1}G$ is the Schur complement. This implies that

$$\begin{pmatrix} F & G \\ D & 0 \end{pmatrix} \begin{pmatrix} F & G \\ 0 & -S \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ DF^{-1} & I \end{pmatrix}, \tag{2.5}$$

which, in turn, suggests a preconditioning strategy for (1.1). If it were possible to use the matrix

$$\mathcal{Q} = \begin{pmatrix} F & G \\ 0 & -S \end{pmatrix} \tag{2.6}$$

as a right-oriented preconditioner, then the preconditioned operator would be the one given in (2.5). All the eigenvalues have the value 1, and it can be shown that this operator contains Jordan blocks of dimension at most 2, and consequently that at most two iterations of a preconditioned GMRES iteration would be needed to solve the system [12].

The key bottleneck in applying $\mathcal{Q}^{-1}$ is the action of the inverse Schur complement operator $(DF^{-1}G)^{-1}$. In practice, this must be replaced with an inexpensive approximation. If it were possible to find a matrix F_p that makes (1.5) an equality, then this would imply that

$$F^{-1}G = GF_p^{-1}, \tag{2.7}$$

which in turn would mean $S = \tilde{S}$ in (1.4). In general, (1.5) is an over-determined system (with $range(G)$ a proper subset of $range(F)$), so it is not possible to find such an F_p. Consequently, we have an approximation in (1.4). Using $\tilde{S} = (DG)F_p^{-1}$ within a preconditioning operation entails applying the action of the inverse, given by

$$\tilde{S}^{-1} = F_p(DG)^{-1}. \tag{2.8}$$

Note that DG is a scaled discrete Laplacian operator, so that working with (2.8) will require a Poisson solve, followed by a matrix-vector product with F_p. For purposes of efficiency, the action of $(DG)^{-1}$ is often approximated by an inexact Poisson solver without any significant degradation in convergence. Thus, application of (2.8) is inexpensive.

A good approximation can be expected in (1.5) because of the nature of the differential operators associated with the matrices. In particular, suppose the convection–diffusion operator

$$(-\nu\nabla^2 + (\mathbf{w}\cdot\text{grad}))_p, \tag{2.9}$$

can be defined on the pressure space (where the subscript p indicates that the operator is acting on the pressure space). Consider the commutator with the gradient operator,

$$(-\nu\nabla^2 + (\mathbf{w}\cdot\text{grad}))\,\text{grad} - \text{grad}(-\nu\nabla^2 + (\mathbf{w}\cdot\text{grad}))_p\,. \tag{2.10}$$

If $\mathbf{w}$ is constant, then this expression would be zero on the interior of Ω, and it is also reasonable to expect it to be small for smooth $\mathbf{w}$. This suggests that if F_p is a discretization of (2.9), then the discrete commutator

$$FG - GF_p \tag{2.11}$$

will also be small. A similar statement would apply if time derivatives are present: as in (2.3), discretization in time would add terms essentially of the form $\frac{1}{\Delta t}I$ to both convection–diffusion operators in (2.10).

In [11],[16], F_p is obtained from discretization of the convection–diffusion operator of (2.9). In this paper, we develop an automated construction of F_p designed explicitly to make the commutator (1.5) small. This removes the process of discretization from the design of the iterative solution algorithm and enables the preconditioning to adapt automatically to different discretizations without any intervention on the part of the designer of a Navier–Stokes code. One of the ideas for approximating (1.5) is to adapt methods for computing sparse approximate inverses to define sparse approximate commutators. This will be considered in Section 4.

Before exploring sparse approximations, it is worthwhile to consider a classical normal equations solution to (1.5). In particular, suppose f_j, the jth column of F_p, is computed to solve the least-squares problem with respect to the Euclidean norm,

$$\min \|Gf_j - [FG]_j\|_2 ,$$

where $[FG]_j$ represents the jth column of the matrix FG. This leads to the normal equations

$$(G^T G)f_j = G^T [FG]_j .$$

An equivalent formulation is that F_p minimizes the Frobenius norm of the error in the complete system

$$\min \|GF_p - FG\|_F . \tag{2.12}$$

Thus, the solution is

$$F_p = (G^T G)^{-1} G^T FG. \tag{2.13}$$

When $D = G^T$ (as is normally the case), the resulting Schur complement preconditioner becomes

$$(G^T G)^{-1} G^T FG(G^T G)^{-1}. \tag{2.14}$$

This corresponds exactly to the *BFBt* preconditioner proposed in [6] and also in [15]. This method can be considered as an alternative to the pressure convection–diffusion preconditioner; it is of particular interest for Newton's method, for which (in contrast to Picard iteration) it is difficult to derive a discretization of (2.9) that will make (2.11) small. We should also point out that the original derivation of the *BFBt* preconditioner is considerably different from the viewpoint given here. The notion of the normal equations solution to (1.5) is in our opinion more intuitive than the description given in [6] and provides additional insight into this preconditioner. From this perspective, however, the approximate solution of the commutator equation is certainly not a new idea, as it has already been done with the *BFBt* preconditioner. In effect, we are proposing alternate solutions and modifications to (1.5) that may have better properties. These modifications will include the use of diagonal scaling in (2.12) as well as boundary condition adjustments. It will be shown how these simple improvements can significantly accelerate convergence. Note also that use of (2.14) will entail two Poisson solves in each invocation of the preconditioner; it will become apparent that sparse approximate inverses can be used to eliminate one of these two solves.

3. Diagonally Scaled Commutators. The construction of an algebraic commutator need not explicitly rely on the PDE analog used to derive the pressure convection–diffusion preconditioning. This implies that alternative relationships derived from commutators can be explored when developing Schur complement approximations. Consider, for example,

$$M_2^{-1} G F_p \approx M_2^{-1} F M_1^{-1} G \tag{3.1}$$

where M_2 and M_1 are diagonal matrices. The operator M_2^{-1} can be thought of as a weight matrix that transforms (2.12) into a weighted least-squares problem. The introduction of M_1^{-1} can be viewed as a way to precondition F and G so that they are more amenable to commuting. After premultiplying (3.1) by $DF^{-1}M_2$, it is easy to show that the inverse Schur complement is now approximated by

$$(DF^{-1}G)^{-1} \approx F_p(DM_1^{-1}G)^{-1}, \tag{3.2}$$

where

$$F_p = (G^T M_2^{-2} G)^{-1} G^T M_2^{-2} F M_1^{-1} G \tag{3.3}$$

when the normal equations are used to produce an approximate solution to (3.1). The net effect of "preconditioning" the commutator equation in this way is that a new definition of F_p is given and the inverse discrete Poisson operator in (2.8) is now replaced by a discrete variable-coefficient diffusion operator (when M_1 is a diagonal matrix), which can still be effectively handled with solution methods such as multigrid. In this paper, we will primarily consider choices for M_1 and M_2 that are based on the diagonal of the velocity mass matrix. However, before exploring this, it is worthwhile to consider another choice for M_1 and M_2 that highlights connections between F_p solvers and the well-known SIMPLE method [13].

3.1. Diagonal Scaling and SIMPLE. One interesting aspect of the diagonally scaled commutator equation is that it bridges the gap between pressure convection–diffusion solvers and traditional pressure-correction methods such as SIMPLE [13]. In particular, the original SIMPLE method is a numerical time marching procedure for incompressible flow in which the inverse Schur complement is approximated by $(D\, diag(F)^{-1}G)^{-1}$, where $diag(F)$ is the diagonal matrix whose entries are taken from the main diagonal of F. When $M_1 = diag(F)$ and F_p is the identity matrix, the right side of (3.2) is identical to SIMPLE's approximation of the inverse Schur complement. In general, F_p will be close to the identity when $F\, diag(F)^{-1}$ is close to the identity. This, in turn, occurs when the entries on the diagonal of F are much larger than the off-diagonal entries, as is often the case for time-dependent problems with small time steps. Thus, we can conclude that the Schur complement approximations for the diagonally scaled commutator and SIMPLE are basically the same in the small time-step scenario.

The true advantage of the F_p solvers occurs for larger time steps or steady-state problems. Let A denote the coefficient matrix of (1.2). Consider the new block preconditioner based on the factorization

$$\tilde{A} \equiv \begin{pmatrix} F & 0 \\ D & -D\, diag(F)^{-1}G \end{pmatrix} \begin{pmatrix} I & diag(F)^{-1}G \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & F_p^{-1} \end{pmatrix}. \tag{3.4}$$

This new preconditioner is similar to standard F_p methods with the exception that the underlying block decomposition is built on SIMPLE's approximate block factorization as opposed to (2.4). It is well known that when F_p is the identity, then the LU decomposition given by (3.4) corresponds to SIMPLE's underlying matrix factorization [14]. By multiplying out matrices, it is easy to show that

$$\tilde{A} = \begin{pmatrix} F & F diag(F)^{-1} G F_p^{-1} \\ D & 0 \end{pmatrix}$$

and that the difference between $\tilde{A}$ and the discrete incompressible Navier–Stokes operator, A, of (1.2) is given by

$$A - \tilde{A} = \begin{pmatrix} 0 & (GF_p - F diag(F)^{-1} G) F_p^{-1} \\ 0 & 0 \end{pmatrix}. \tag{3.5}$$

It follows once again that the SIMPLE method (where F_p is the identity) will perform well when $G - F diag(F)^{-1} G$ is small, i.e., when F has relatively large diagonal entries. On the other hand, comparison of the structure of the nonzero block in the $(1,2)$ entry of (3.5) with (3.1), the "preconditioned commutator relation" using $M_1 = diag(F)$ and $M_2 = I$, indicates that the error matrix $A - \tilde{A}$ will be small if the commutator is small, irrespective of the size of the diagonal of F.

3.2. Diagonal Scaling and the Mass Matrix. Replacement of a differential commuting relationship with an algebraic commuting relationship assumes certain additional properties of the discrete operator. If, for example, a finite difference approach such as the marker-and-cell method [10] is used to discretize the Navier–Stokes equations, then the relation (2.11) looks very much like its continuous analogue (2.10), and heuristic justification for the validity of (2.11) comes from Taylor series arguments. On the other hand, for mixed finite element discretizations with different order elements, such as the stable Q_2–Q_1 (biquadratic velocity, bilinear pressure) element, the discrete commutator (2.11) will typically not be small. This does not present a problem when the required matrix F_p is obtained as in [11, 16] by constructing a discrete convection–diffusion operator using the given pressure discretization (bilinear elements in the example noted above). However, it complicates the construction of F_p when a purely algebraic approach is used.

To gain insight into this matter, consider the differential operators

$$\mathcal{L}_1 u \equiv -u_{xx} \qquad \text{and} \qquad \mathcal{L}_2 u \equiv u_x$$

on the one-dimensional domain $0 \le x \le 1$, with periodic boundary conditions. Clearly, the differential operators formally commute. That is, $\mathcal{L}_1\mathcal{L}_2$ is equivalent to $\mathcal{L}_2\mathcal{L}_1$. Now let L_1 and L_2 be the discrete approximations to $\mathcal{L}_1$ and $\mathcal{L}_2$ obtained from piecewise linear nodal finite elements

$$\phi_i(x) = \begin{cases} \alpha_i \frac{x - x_{i-1}}{x_i - x_{i-1}} & x_{i-1} \le x \le x_i \\ \alpha_i \frac{x - x_{i+1}}{x_i - x_{i+1}} & x_i \quad \le x \le x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

and a Galerkin discretization. Here x_i denotes the location of the ith mesh point and α_i effectively scales each element in an arbitrary way. Standard nodal elements have $\alpha_i = 1$ for all i, and it can be shown that on uniform meshes

$$L_1^s L_2^s = L_2^s L_1^s \,,$$

where the superscript 's' indicates standard (or unscaled) nodal element discretization. The use of arbitrary scaling, however, effectively scales the corresponding discrete operators:

$$L_1 = D_\alpha L_1^s D_\alpha \qquad \text{and} \qquad L_2 = D_\alpha L_2^s D_\alpha$$

where L_1 and L_2 denote discretizations generated with arbitrary scaling and D_α is the diagonal matrix with α_i in the (i,i)th entry. Although L_1 and L_2 do not commute for arbitrary scaling, we do have

$$L_1 D_\alpha^{-2} L_2 = L_2 D_\alpha^{-2} L_1. \tag{3.6}$$

That is, an appropriate scaling enables commutativity.

The example above is artificial, but an analogous strategy can be used to enhance performance in the setting we are concerned with. In particular, it is possible to choose a diagonal matrix M_1 so that a version of the commutator is small on some components of the pressure space. For the Q_2–Q_1 discretization, if M_1 is chosen to be $diag(M)$, the diagonal matrix whose entries are those on the diagonal of the velocity mass matrix M, then $M_1^{-1}Gq$ is constant on grid points at the interior of Ω whenever the nodal vector q comes from a linear function. Moreover, for such q and this choice of M_1, if F_p is a discrete bilinear approximation to (2.9) with constant $\mathbf{w}$, then

$$(GF_p - FM_1^{-1}G)q = 0 \quad \text{in the interior of } \Omega.$$

Thus, we use this strategy for choosing M_1. Note that in the example leading to (3.6), the diagonal of the mass matrix is simply D_α^2, so that this choice of M_1 is consistent with the motivating example.

We will derive an automated version of the pressure convection–diffusion preconditioner in the following section. We conclude here by presenting a variant of the *BFBt* method based on this approach. Insertion of the normal equations solution to (3.1) in (3.2) yields the following approximation to the inverse Schur complement (assuming $D = G^T$):

$$(DF^{-1}G)^{-1} \approx (G^T M_2^{-2} G)^{-1} (G^T M_2^{-2} F M_1^{-1} G)\, (G^T M_1^{-1} G)^{-1}.$$

As noted above, M_1 is the diagonal of the velocity mass matrix. We take $M_2 = sqrt(M_1)$ to make the two variable-coefficient Poisson operators identical. The resulting preconditioner, with these choices for M_1 and M_2, will be referred to as the *scaled BFBt* method. It can be also be shown that the matrix F_p obtained from the normal equations for (3.1) corresponds to a convection–diffusion operator in (2.9) for which the L_2-norm of the commutator (2.10), viewed as an operator on the (finite element) pressure space, is approximately minimized; see [8, Chapter 8] for details. As will be seen in Section 5, the introduction of this diagonal scaling can have a dramatic impact on the overall convergence of the method.

4. Sparse Approximate Commutator (SPAC). One disadvantage of both the *BFBt* and scaled *BFBt* methods is that they require two Poisson-like solves during each invocation, as opposed to the single solve needed by the pressure convection–diffusion preconditioners. This implies that the application of the approximate inverse Schur complement is more costly for *BFBt* methods than for pressure convection–diffusion methods.

To alleviate the burden of the additional solve required by *BFBt* methods, we consider approximate solutions to (1.5) built on concepts from sparse approximate inverses. The resulting method will still only require one invocation of the Poisson solver per iteration, although there is an additional setup cost associated with building the sparse approximate inverse. Since sparse approximate inverses are typically used to approximate the matrix inverse of a square matrix, an adaptation is considered that is suitable for the rectangular matrix associated with the approximate commutator equation. We begin by discussing standard sparse approximate inverses.

Sparse approximate inverses were originally developed as preconditioning operators for sparse matrices; they were designed to be easier to compute and use on parallel architectures than other preconditioners such as incomplete LU factorizations. Given a large, sparse matrix A, these methods produce a sparse matrix Q that approximates A^{-1}. The sparse approximate inverse is inherently parallel and intended to retain the convergence properties of an incomplete LU factorization. In addition, sparse approximate inverses are important for situations in which incomplete LU factorizations have difficulties, such as when the factorization encounters a zero pivot or produces unstable factors [5]. Let A represent a generic matrix of order n, and let I denote the identity matrix of order n. Typically, sparse approximate inverse techniques are based on producing a sparse matrix Q that minimizes the Frobenius norm of the residual matrix

$$\min_{Q} \|AQ - I\|_F, \tag{4.1}$$

which was first suggested by Benson and Frederickson [1, 2]. The residual matrix minimization reduces to n independent least-squares problems

$$\min_{q_j} \|Aq_j - e_j\|_2, \quad j = 1, \ldots, n, \tag{4.2}$$

where q_j is the jth column of Q, and e_j is the jth unit vector, i.e., the jth column of I.

Note that our task is less difficult than that required for construction of sparse approximate inverses. The key obstacle in the latter setting is picking the proper sparsity pattern of Q to capture enough of A^{-1}, which is usually dense even when A is sparse. In our case, it is reasonable to expect that the matrix F_p needed in the discrete commutator equation is sparse (as we know this to be the case in standard pressure convection–diffusion preconditioning). Thus we have the simpler task of computing a sparse approximation to a sparse matrix.

There are several strategies for computing sparse approximate inverses (see, e.g., [3, §5]). Many sparse approximate inverse methods (e.g., Chow [4]) require iterated applications of matrix-vector products of the form $Ar, A(Ar), \ldots$. The SPAI algorithm of Grote and Huckle [9] avoids this construction and is natural to apply in the present setting where a commutator is required for a rectangular matrix. We call this adaptation *SPAC*. Before discussing the adaptation of SPAI to commutators, it is important to note that approximate inverses based on square matrices can also be used by instead considering the normal equations

$$(G^T M_2^{-2} G) F_p = G^T M_2^{-2} F M_1^{-1} G.$$

In this case, one replaces (4.1) by

$$\min_{F_p} \|G^T M_2^{-2} G F_p - G^T M_2^{-2} F M_1^{-1} G\|_F^2,$$

which involves square matrices. We do not pursue this further but instead consider ideas for computing sparse matrices that directly address the commutator relationship.

SPAC is based on replacing (4.1) with

$$\min_{F_p \text{ sparse}} \|\tilde{G} F_p - \tilde{F} \tilde{G}\|_F^2, \tag{4.3}$$

where $\tilde{G} = M_2^{-1} G$ and $\tilde{F} = M_2^{-1} F M_1^{-1} M_2$. This reduces to a set of n constrained least-squares problems:

$$\min_{f_j \text{ sparse}} \|\tilde{G} f_j - b_j\|_2^2, \tag{4.4}$$

where b_j is the jth column of $\tilde{F}\tilde{G}$ and f_j is the jth column of F_p.

Our sparse approximate commutator algorithm (*SPAC*) follows the technique of Grote and Huckle [9]. We will use the following notation: given index sets $\mathcal{I}$, $\mathcal{J}$, let $\tilde{G}(\mathcal{I},\mathcal{J})$ denote the submatrix of $\tilde{G}$ with entries $\tilde{g}_{ij}$ where $i \in \mathcal{I}$, $j \in \mathcal{J}$; by analogy with MATLAB, a colon ":" represents the complete index set $\{1,\ldots,n\}$. Similarly, for a vector v, $v(\mathcal{I})$ or $v(\mathcal{J})$ denotes the subvector of v with entries v_i or v_j where $i \in \mathcal{I}$ and $j \in \mathcal{J}$. As in [9], we describe the algorithm for one column of F_p. We begin by getting an initial sparsity pattern. Let $\mathcal{B}$ be the set of indices of the nonzero rows in b_j, let $\mathcal{J}$ be the set of indices of nonzero columns of $\tilde{G}(\mathcal{B},:)$, and let $\mathcal{I}$ be the set of indices of the nonzero rows of $\tilde{G}(:,\mathcal{J})$. The algorithm starts by solving the $n_1 \times n_2$ least-squares problem $\tilde{G}(\mathcal{I},\mathcal{J})f_j(\mathcal{J}) = b_j(\mathcal{I})$, where $n_1 = |\mathcal{I}|$ and $n_2 = |\mathcal{J}|$. The vector f_j obtained by inserting $f_j(\mathcal{J})$ into the zero vector is the initial approximation to the jth column of F_p. We improve F_p by augmenting the sparsity structure to obtain a more effective commutator. As in [9], we accomplish this by reducing the current error $\|\tilde{G}F_p - \tilde{F}\tilde{G}\|_F^2$, i.e., reducing $\|\tilde{G}f_j - b_j\|_2^2$ for each column $j = 1,\ldots,n$ of F_p. Let the residual $r = \tilde{G}f_j - b_j$. If $r = 0$, then f_j is exactly the jth column of the commutator and no improvement is possible. If $r \neq 0$, we augment the sets of indices $\mathcal{I}$ and $\mathcal{J}$ to reduce $\|r\|_2$. Let $\mathcal{R}$ be the set of indices of the nonzero elements of r, and let $\tilde{\mathcal{J}}$ be the set of new column indices that appear in $\mathcal{R}$ but not in $\mathcal{J}$. To determine the most profitable reduction in $\|r\|_2$, for each $k \in \tilde{\mathcal{J}}$ we solve the one-dimensional minimization problem

$$\min_{\mu_k} \|r + \mu_k \tilde{G}(:,\tilde{\mathcal{J}}(k))\|_2, \tag{4.5}$$

whose solution is given by

$$\mu_k = -\frac{(r^T\tilde{G}(:,\tilde{\mathcal{J}}(k)))^2}{\|\tilde{G}(:,\tilde{\mathcal{J}}(k))\|_2^2}, \tag{4.6}$$

where $\tilde{\mathcal{J}}(k)$ is the kth element of the index set $\tilde{\mathcal{J}}$. For each k, we compute $\rho_k = \|r + \mu_k\tilde{G}(:,\tilde{\mathcal{J}}(k))\|_2^2$, the square of the Euclidean norm of the new residual, using

$$\rho_k = \|r\|_2^2 - \frac{(r^T\tilde{G}(:,\tilde{\mathcal{J}}(k)))^2}{\|\tilde{G}(:,\tilde{\mathcal{J}}(k))\|_2^2}. \tag{4.7}$$

We reduce $\tilde{\mathcal{J}}$ to the set of indices with the lowest ρ_k. In particular, we delete from $\tilde{\mathcal{J}}$ the indices whose ρ values are greater than the median of the ρ_k. We improve $\mathcal{J}$ by augmenting it with $\tilde{\mathcal{J}}$, $\mathcal{J} = \mathcal{J} \cup \tilde{\mathcal{J}}$. $\mathcal{I}$ is updated to contain the indices of the nonzero rows of $\tilde{G}(:,\mathcal{J})$. We then repeat the solution of the least-squares problem $\tilde{G}(\mathcal{I},\mathcal{J})f_j(\mathcal{J}) = b_j(\mathcal{I})$, updating the elements of f_j indexed by $\mathcal{J}$ to produce an improved column of F_p. To reduce fill, we drop entries of f_j that are less than a given tolerance.

We stop improving $\mathcal{J}$ and the current column of the commutator F_p when we have achieved sufficient reduction in the error $\|\tilde{G}F_p - \tilde{F}\tilde{G}\|_F^2$. The entire algorithm is given below.

Algorithm 1 *SPAC*

```
For each column $b_j$ of $\tilde{F}\tilde{G}$:
    • Find initial index sets of nonzero rows & columns $(\mathcal{I},\mathcal{J})$ of $\tilde{G}$
        – Let $\mathcal{B}$ = indices of nonzero rows in $b_j$
    – Let $\mathcal{J}$ = indices of nonzero columns of $\tilde{G}(\mathcal{B},:)$
    – Let $\mathcal{I}$ = indices of nonzero rows of $\tilde{G}(:,\mathcal{J})$
• Set $r_{norm} = 1$
• Set $reduction = 1$
• while( $reduction/r_{norm} > tol_1$ & $r_{norm} > tol_2$ )
    – Let $\mathcal{J}_{prev} = \mathcal{J}$
    – Solve the least-squares problem $\min \|\tilde{G}(\mathcal{I},\mathcal{J})f_j(\mathcal{J}) - b_j(\mathcal{I})\|_2^2$ for $f_j(\mathcal{J})$
      updating the elements of $f_j$ indexed by $\mathcal{J}$
    – Compute the residual $r = \tilde{G}f_j - b_j$
    – Set $r_{norm} = \|r\|_2$
    – Let $\mathcal{R}$ = indices of nonzero entries of $r$
    – Let $\tilde{\mathcal{J}}$ = the set of new column indices of $\tilde{G}$ that appear in all $\mathcal{R}$ rows
      but not in $\mathcal{J}$
    – For each $k = 1, \ldots, |\tilde{\mathcal{J}}|$
        ∗ Compute $\mu_k = (r^T\tilde{G}(:,\tilde{\mathcal{J}}(k)))^2/(\tilde{G}(:,\tilde{\mathcal{J}}(k))^T\tilde{G}(:,\tilde{\mathcal{J}}(k)))$
        ∗ Set $\rho_k = \|r\|_2^2 - \mu_k$
    – Set $reduction = \max_k \mu_k$
    – Improve $\mathcal{J}$ by taking the $\tilde{\mathcal{J}}$ indices with the smallest $\rho$. Delete from $\tilde{\mathcal{J}}$
      the indices whose $\rho$ value is greater than the median value of the $\rho_k$. Let
      $\mathcal{J} = \mathcal{J} \cup \tilde{\mathcal{J}}$
    – Improve $\mathcal{I}$ by setting $\mathcal{I}$ = nonzero rows of $\tilde{G}(:,\mathcal{J})$
• Drop entries of $f_j < drop\_tol$. Let $ind$ = indices of the remaining elements
• Set $F_p(\mathcal{J}_{prev}(ind), j) = f_j(ind)$
```

The resulting *SPAC* algorithm is almost identical to the method presented by Grote and Huckle. Therefore, its cost should be similar to this sparse approximate inverse algorithm. It is also important to keep in mind that the dimension of the Schur complement system is only a small fraction of the entire system (often about one quarter for three dimensional problems) and that fairly sparse approximations can be used (as F_p in pressure convection–diffusion preconditioning is sparse). Thus, it is natural to expect that the cost to construct the *SPAC* operator (relative to the cost of solving the entire saddle point system) is considerably less than the setup costs associated with standard sparse approximate inverses.

5. Results. Numerical experiments are presented for two benchmark problems in two dimensions, models of flow over a backward facing step and in a lid driven cavity application. The linear problems were generated by applying a Picard iteration to the nonlinear system and stopping this iteration when the nonlinear residual satisfied

$$\left\| \begin{pmatrix} \mathbf{f} - \left(\mathbf{F}(\mathbf{u}^{(m)})\mathbf{u}^{(m)} + Gp^{(m)}\right) \\ g - D\mathbf{u}^{(m)} \end{pmatrix} \right\|_2 \leq 10^{-5} \left\| \begin{pmatrix} \mathbf{f} \\ g \end{pmatrix} \right\|_2, \tag{5.1}$$

where $\mathbf{F}(\mathbf{u})\mathbf{u}$ is the discretization of the nonlinear convection–diffusion operator, $-\nu\nabla^2\mathbf{u} + (\mathbf{u}\cdot\text{grad})\mathbf{u}$, and g comes from boundary data. At each Picard step m, the Oseen equations (1.2) are solved for a correction to the current iterate; the right hand side of the linear system is the nonlinear residual given on the left of (5.1). The numbers reported correspond to the total number of GMRES iterations (without restarting) required to solve the saddle point system associated with the last Picard iteration, using various preconditioners. The linear iteration was stopped when the residual of the linear system (1.2) satisfied

$$\|\mathbf{r}^{(k)}\|_2 \leq 10^{-6}\|\mathbf{f}^{(m)}\|_2 \tag{5.2}$$

starting with a zero initial iterate, where $\mathbf{f}^{(m)}$ is the current nonlinear residual vector and $\mathbf{r}^{(k)}$ is the residual at the kth linear iteration. The performance of the iterative solvers was not sensitive to the particular Picard step, and these results are representative.

All the preconditioners are of the form given by (2.6) where an exact solver is used for all needed Poisson solves and convection–diffusion solves. The inverse Schur complement is the only aspect that is varied; preconditioners tested include an application-provided pressure convection–diffusion preconditioner (i.e., the method of [11, 16], denoted "F_p" in the tables), *BFBt*, scaled *BFBt* (denoted *Sc-BFBt*), *SPAC* F_p without diagonal scaling, and *SPAC–M* F_p (*SPAC* using the same diagonal scaling as scaled *BFBt* to define $\tilde{F}$ and $\tilde{G}$). For the application-provided pressure convection–diffusion preconditioner, the matrix F_p is obtained from the convection–diffusion operator that determines F, discretized on the pressure space. Two discretizations were considered: the Q_2–Q_1 (biquadratic velocity / bilinear pressure) mixed finite element discretization [8], and the marker-and-cell (MAC) finite difference discretization [10]. The experiments were done in MATLAB. The Q_2–Q_1 discretization and all tests of iterative methods were performed using the Incompressible Flow Iterative Solution Software package IFISS [7] developed in conjunction with the volume [8].

First, we compare methods on the backward facing step. In this example, a Poiseuille flow profile (steady horizontal flow in a channel driven by pressure difference between the two ends) is imposed on the inflow boundary ($x = -1; 0 \leq y \leq 1$), and a no-flow (zero velocity) condition is imposed on the walls. At the outflow boundary ($x = 5; -1 < y < 1$), the Neumann condition

$$\begin{array}{rcl} \nu \frac{\partial u_x}{\partial x} - p & = & 0 \\ \frac{\partial u_y}{\partial x} & = & 0 \end{array} \tag{5.3}$$

is satisfied and automatically sets the mean outflow pressure to zero [8]. (Here the velocity vector is $\mathbf{u} = (u_x, u_y)^T$.) The Q_2–Q_1 discretization is performed on a uniform finite element grid, where the grid is set up so that the rectangle enclosing the step would contain a uniform $n \times 3n$ grid of velocity nodes of width $h = 2/n$. Figure 5.1 shows the velocity field and pressure field for a representative solution to a two-dimensional backward facing step problem with $n = 64$.

In Table 5.1 and Table 5.2, the number of GMRES iterations needed to satisfy the stopping criterion (5.2) are shown for a variety of Reynolds numbers $Re = 2/\nu$, where the viscosity ν is as in (1.1). For the *SPAC*-generated F_p (both diagonally scaled and unscaled), we use the following drop tolerances in Algorithm 1: $drop\_tol = 0.1, tol_1 = 0.01$, and $tol_2 = 0.1$.

Re	F_p	*BFBt*	Sc-*BFBt*	*SPAC*	*SPAC-M*
10	30	54	19	55	23
100	42	64	21	76	30
200	47	65	22	86	41

TABLE 5.1
Preconditioned GMRES iterations for backward facing step problem with an underlying 64×192 *grid,* Q_2–Q_1 *discretization.*

As the tables illustrate, the standard *BFBt* method requires noticeably more iterations than the application-provided pressure convection–diffusion method. This type of behavior has been observed by others and has lead to the perception that the

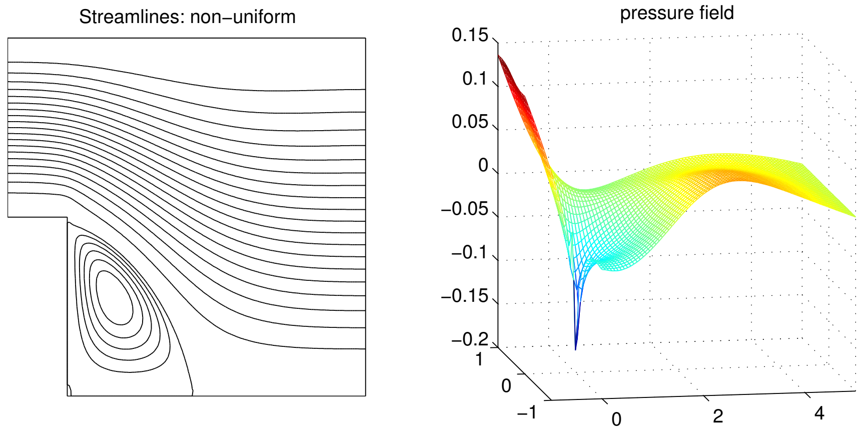

FIG. 5.1. *Streamline contours (left) and pressure plot (right) of a Q_2–Q_1 approximation to a backward facing step problem with parabolic inflow boundary conditions and natural outflow boundary conditions with $Re = 100$ on an underlying 64×192 grid.*

BFBt method is largely inferior to the pressure convection–diffusion method. However, inclusion of diagonal scaling improves the method so that it now significantly outperforms the pressure convection–diffusion method (often achieving a factor of two improvement in the number of iterations for higher Reynolds numbers). The tables further illustrate how the *SPAC* method somewhat mirrors the behavior of *BFBt*. In particular, the non-scaled version does not perform well while the scaled version is quite competitive with the application-provided pressure convection–diffusion preconditioner. It is also worth noting that these *SPAC* iterations correspond to relatively large drop tolerances. We have performed additional experiments with smaller tolerances and have noticed only modest reductions in iteration counts. (See also some comments below on use of *larger* tolerances.) It should be kept in mind that in the limit as the drop tolerance goes to zero, the *SPAC* methods become identical to the *BFBt* methods. Of course, in this limit the computation of the sparse approximate commutator would be quite costly compared to one using a large tolerance.

Re	F_p	*BFBt*	Sc-*BFBt*	*SPAC*	*SPAC–M*
10	33	82	23	83	32
100	58	104	29	124	39
200	63	106	29	142	60

TABLE 5.2
Preconditioned GMRES iterations for backward facing step problem with an underlying 128×384 *grid,* Q_2–Q_1 *discretization.*

Next, we compare preconditioners on the lid driven cavity in two dimensions using a Q_2–Q_1 discretization. Specifically, we consider a square region with unit length sides. Velocities are zero on all edges except the top (lid), which has a driving velocity of one. The two-dimensional lid driven cavity is a well-known benchmark for fluids problems and contains many features of harder flows. Figure 5.2 shows a representative solution.

In Table 5.3 and Table 5.4, the number of iterations needed to reduce the initial residual by 10^6 are shown for a variety of Reynolds numbers $Re = 2/\nu$; the *SPAC*

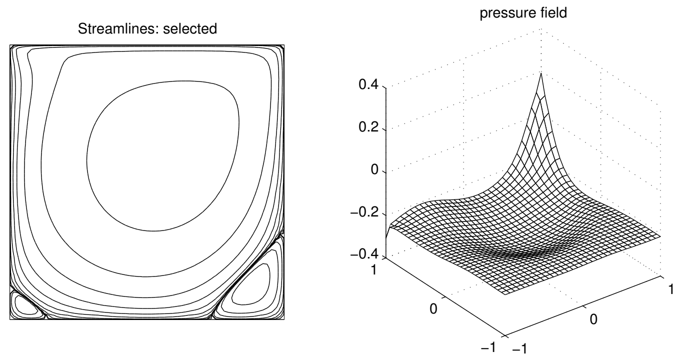

FIG. 5.2. *Exponentially spaced streamline contours (left) and pressure plot (right) of a* Q_2*–*Q_1 *approximation to a 2D lid driven cavity problem with* $Re = 500$ *on a* 64×64 *grid.*

drop tolerances are identical to those used for the backward facing step.

Re	F_p	*BFBt*	Sc-*BFBt*	*SPAC*	*SPAC–M*
100	27	46	21	49	21
500	47	77	34	81	38
1000	70	93	55	98	63
2000	130	131	110	132	119

TABLE 5.3
GMRES iterations for lid driven cavity problem on a 64×64 *grid,* Q_2*–*Q_1 *discretization.*

Re	F_p	*BFBt*	Sc-*BFBt*	*SPAC*	*SPAC–M*
100	27	68	27	73	27
500	43	106	37	127	42
1000	56	125	45	149	65
2000	106	154	85	192	105

TABLE 5.4
GMRES iterations for lid driven cavity problem on a 128×128 *grid,* Q_2*–*Q_1 *discretization.*

In this case, the *BFBt* results are better relative to the pressure convection–diffusion method than in the case of the backward facing step. Once again the use of diagonal scaling improves the *BFBt* method so that it outperforms the pressure convection–diffusion method (though not as dramatically as in the backward facing step problem). Similar to the results for the backward facing step, *SPAC* and *SPAC–M* roughly mirror their *BFBt* counterparts, and the scaled algebraic methods compare quite well with results for the application-provided pressure convection–diffusion preconditioner. We believe that in the large Reynolds number case, lack of accuracy of the discrete convection–diffusion operators F and F_p hinders the effectiveness of the pressure convection–diffusion preconditioner; comparison of Tables 5.3 and 5.4 shows that performance of the pressure convection–diffusion preconditioner and of the scaled algebraic methods improves as the mesh is refined. This is in contrast to the backward facing step problem where all the preconditioners exhibit some iteration growth. We suspect that iteration growth on the backward facing step problem (for all the preconditioners) is associated with the boundary conditions of the F_p system. This will be explored in our final example.

Next, we verify that similar convergence behavior is exhibited for different discretizations. Specifically, iterations for a lid driven cavity problem with a MAC discretization are given in Table 5.5 and Table 5.6. (Here the problem is posed on $[0,1] \times [0,1]$ and $Re = 1/\nu$.) In this case, scaled versions of *BFBt* and *SPAC* are not

Re	F_p	*BFBt*	*SPAC*	*SPACbc*
10	15	18	21	15
100	26	25	30	23
1000	54	42	51	50

Table 5.5

Preconditioned GMRES iterations for lid driven cavity problem on a 64×64 *grid, MAC discretization.*

Re	F_p	*BFBt*	*SPAC*	*SPACbc*
10	16	22	27	15
100	26	32	39	24
1000	57	53	65	54

Table 5.6

Preconditioned GMRES iterations for lid driven cavity problem on a 128×128 *grid, MAC discretization.*

considered, as all the velocity basis functions effectively have the same scaling. That is, the equivalent of the mass matrix for the MAC scheme has a constant diagonal and so the introduction of M_1 and M_2 does not change the method. In the last column of these tables, we consider a further modification to the matrix F_p produced by *SPAC* so that the boundary conditions correspond to those in the application-provided version of F_p. In particular, for problems such as the lid-driven cavity with enclosed flow, it is standard for the application-provided method to enforce Neumann boundary conditions when discretizing the convection–diffusion operator on the pressure space. For problems such as the backward facing step where both inflow and outflow conditions are present the situation is somewhat more complicated [8]. Unfortunately, the *SPAC* preconditioner does not automatically reproduce the proper boundary conditions. This should not be surprising as the notion of commuting does not apply at the boundaries. This implies that the algebraic commuting equation is not satisfied near boundaries and so the computed F_p stencil at the boundaries may not be best. To mimic Neumann boundary conditions, the diagonal of the *SPAC* F_p is modified so that the row sums are identically zero.

Once again the results indicate that both the *BFBt* and *SPAC* preconditioners are competitive with the application-provided pressure convection–diffusion preconditioner. Further, the inclusion of the boundary modification to *SPAC*'s F_p yields an additional reduction in the number of required iterations, with boundary-modified *SPAC* performing better than the application-provided version of the pressure convection–diffusion method. More importantly, the *SPACbc* method appears to exhibit mesh independent convergence while mesh independence is less clear for the other two algebraic algorithms. Given this impact on convergence, further study of boundary conditions within these preconditioners needs to be explored.

We conclude this section with some observations on the costs, in CPU time, of the *SPAC* preconditioning. It has been observed [4] that the setup cost may be significant when computing a sparse approximate inverse with a dynamically determined sparsity pattern. We have observed a similar phenomenon in computing *SPAC*. As with standard sparse approximate inverses, the cost can be significantly reduced for the *SPAC* algorithm if a predetermined sparsity pattern is employed. As we recall from Section 4, the basic computational task involves the solution of the least-squares problem $\min \|\tilde{G}(\mathcal{I},\mathcal{J})f_j(\mathcal{J}) - b_j(\mathcal{I})\|_2^2$, where b_j is the jth column of $\tilde{F}\tilde{G}$ and f_j is the jth column of F_p to be computed. Instead of dynamically determining the index sets, $\mathcal{I}$ and $\mathcal{J}$, as in Section 4, we can simply define $\mathcal{I}$ to be the set of nonzero row indices in b_j and $\mathcal{J}$ to be the set of nonzero column indices in $G(\mathcal{I},:)$. The least-squares problem is then solved *without* any additional computation to reduce the size of the residual commutator. This algorithm may not lead to as accurate a construction of the commutator, but it tends to perform almost as well and is significantly cheaper than the construction given in Algorithm 1.

Table 5.7 shows the iteration counts and timings using the simplified version of the *SPACbc* preconditioner for the driven cavity problem using a MAC discretization on a 128×128 grid. For the sake of comparison, timings for the application provided pressure convection–diffusion preconditioner are also provided. These data were obtained in serial on a 3.20GHz Intel Xeon processor using the GMRES algorithm provided by MATLAB; the factors of F and G^TG used by the preconditioners were computed once and used throughout the iterations. The *SPACbc* iteration counts should be compared with those of Table 5.6. It is evident that this variant of the *SPAC* preconditioner is competitive with that defined by Algorithm 1 (in terms of iteration counts), and the overhead associated with computing the preconditioner is small.

	F_p		*SPACbc*		
Re	iters	time	build time	iters	time
10	16	37.40	7.17	16	37.46
100	26	60.56	7.45	25	58.53
1000	57	131.61	6.90	55	127.42

TABLE 5.7
Comparison of CPU times and iterations for simplified SPAC *preconditioning, on a* 128×128 *grid, MAC discretization.*

6. Conclusions. Variants of the pressure convection–diffusion preconditioning have been considered for the solution of the linear systems associated with the incompressible Navier–Stokes equations. Standard versions of this method require users to provide a discretization of a convection–diffusion equation over the pressure space. Although kernels needed for this discretization are available within most application codes, the additional burden of supplying this somewhat unnatural operator, denoted F_p, can be quite cumbersome. As an alternative, we have considered the use of an algebraic commuting relationship to determine the F_p operator automatically. We have shown that automatic techniques for generating F_p compete quite well with application-provided pressure convection–diffusion methods in terms of convergence rates. These automatic methods are based on solving an algebraic commuting equation in a least-squares sense.

Two solutions strategies to the commuting equation have been considered. The first centers on the normal equations and actually corresponds to a new interpretation of the *BFBt* method. This new interpretation is somewhat more natural than that given in the original paper proposing the *BFBt* method. Further, this new perspective leads to a diagonally scaled variant of the *BFBt* preconditioner. This diagonal scaling is based on the diagonal of the mass matrix associated with the velocity equations and yields significantly improved convergence rates for the method. Overall, the diagonally scaled *BFBt* method converges in a similar fashion to the application-provided pressure convection–diffusion method and actually outperforms it many times. It should of course be kept in mind that applying a *BFBt* method is more costly than an application-provided pressure convection–diffusion method as two Poisson solves and one convection–diffusion solve are required for each preconditioner invocation as opposed to only one Poisson solve and one convection–diffusion solve for the pressure convection–diffusion preconditioning. The significance of this extra Poisson solve is often quite modest, however, as the convection–diffusion system is much larger than the Poisson system and frequently dominates the run time.

The second solution strategy for the commuting equation builds on sparse approximate inverse techniques. Each column of F_p is built independent of every other column. The basic idea is to identify a sparsity pattern within each column and solve a local least-squares problem to minimize the error in the residual of the commuting equation. To some extent, this method can be viewed as an approximation of the *BFBt* methods. The main advantage of the sparse approximate commutator, *SPAC*, computed in this way is that it requires one Poisson solve per preconditioner invocation similar to the application-provided pressure convection–diffusion methods. Thus, the cost per iteration should be nearly identical for the *SPAC* and the application-provided pressure convection–diffusion methods. Numerical results have been given to illustrate how this method also competes quite effectively with application-provided pressure convection–diffusion preconditioners.

REFERENCES

[1] M. W. Benson, *Iterative solution of large scale linear systems*, MSc thesis, Lakehead University, Thunder Bay, Ontario, 1973.

[2] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Math., 22 (1982), pp. 127–140.

[3] M. Benzi, *Preconditioning techniques for large linear systems: a survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[4] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.

[5] E. Chow and Y. Saad, *Approximate inverse preconditioners for general sparse matrices*, Tech. Rep. UMSI 94/101, Minnesota Supercomputer Institute, University of Minnesota, 1994.

[6] H. C. Elman, *Preconditioning for the steady-state Navier–Stokes equations with low viscosity*, SIAM J. Sci. Comput., 20 (1999), pp. 1299–1316.

[7] H. C. Elman, A. R. Ramage, and D. J. Silvester, *Incompressible Flow Iterative Solution Software Package*. http://www.cs.umd.edu/∼elman/ifiss/.

[8] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers*, Oxford University Press, Oxford, 2005. To appear.

[9] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.

[10] F. H. Harlow and J. E. Welch, *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, The Physics of Fluids, 8 (1965), pp. 2182–2189.

[11] D. Kay, D. Loghin, and A. Wathen, *A preconditioner for the steady-state Navier–Stokes equations*, SIAM J. Sci. Comput., 24 (2002), pp. 237–256.

[12] M. F. Murphy, G. H. Golub, and A. J. Wathen, *A note on preconditioning for indefinite linear systems*, SIAM J. Sci. Comput., 21 (2000), pp. 1969–1972.

[13] S. V. Patankar and D. A. Spalding, *A calculation procedure for heat, mass and momentum*

transfer in three dimensional parabolic flows, Int. J. Heat and Mass Trans., 15 (1972), pp. 1787–1806.

[14] J. B. Perot, *An analysis of the fractional step method*, J. Comp. Phys., 108 (1993), pp. 51–58.

[15] F. Saleri and A. Veneziani, *Pressure-correction algebraic splitting methods for the incompressible Navier–Stokes equations*, Tech. Rep. MOX-26, Politechnico Milano, 2003.

[16] D. Silvester, H. Elman, D. Kay, and A. Wathen, *Efficient preconditioning of the linearized Navier–Stokes equations for incompressible flow*, J. Comp. Appl. Math., 128 (2001), pp. 261–279.

[17] A. Smith and D. Silvester, *Implicit algorithms and their linearisation for the transient incompressible Navier–Stokes equations*, IMA J. Numer. Anal., 17 (1997), pp. 527–545.

[18] S. Turek, *A comparative study of time-stepping techniques for the incompressible Navier–Stokes equations: from fully implicit non-linear schemes to semi-implicit projection methods*, Int. J. Numer. Meth. Fluids, 22 (1996), pp. 987–1011.